0A 0 03 06 0 0 0 01 02 03 04 05 06 07 08 09 0B 0C 0D 0E 0F 0
0 0 0 0 0 0 01 02 03 04
05 06 07 08 09 0A 0B 0C 0D 0E 0F 0 0 0 0 0
0

Syntactic Codes and Grammar Refinement

Marcus Kracht
II. Mathematisches Institut
Freie Universität Berlin
Arnimallee 3
D-14195 Berlin

Abstract. We call *syntactic coding* a technique which converts syntactic principles or constraints on representations into grammatical rules which can be implemented in any given rule grammar. In this paper we show that any principle or constraint on output trees formalizable in a certain fragment of dynamic logic over trees can be coded in this sense. This allows to reduce in a mechanical fashion most of the current theories of government and binding into GPSG-style grammars. This will be exemplified with Rizzi's *Relativized Minimality.*

Key words: government and binding, relativized minimality, gpsg, dynamic logic

1. Introduction

This paper is an introduction into the method of *syntactic coding.* Syntactic coding is quite simply a method to convert descriptions of syntactic trees or syntactic principles into a rule based grammar – preferrably context-free – which does nothing else but produce exactly those trees which conform to that description. The gain of this method is considerable. First, it is completely algorithmic in nature and so can be implemented in a computer if necessary. Thus the working linguist can concentrate on stating the conditions or principles rather than worry about their possible implementation. Secondly, it can be used to show that certain grammars are context-free just by appealing to the basic properties of this method. Thirdly, it shows quite clearly what powers and limitations of the GPSG grammar style are and in what ways GPSG is related to other grammar systems.

This essay will be rather hardgoing; in order to keep this paper reasonably short we cannot give an introduction into the mathematical tools which we use. Instead I refer to the literature. The techniques used here are from formal language theory, boolean algebra and dynamic logic. For formal languages consult any book, for example (Harrison, 1978), for boolean algebras (Davey and Priestley, 1991) and for dynamic logic consult (Harel, 1984).

I wish to thank Noam Chomsky, Axel Kleinlein and an anonymous referee for helpful discussions and Patrick Blackburn for suggesting to me to write this paper.

2. Technical Preparations

2.1. Boolean Algebras

The two main ingredients of this paper are *boolean algebras* and *context-free grammars*. We assume that the reader has standard knowledge of them. Here we will only introduce some notation and also lesser known techniques. Throughout this paper, all objects are assumed finite, although most methods have an infinite analogue.

A **boolean algebra** is an object $\mathfrak{B} = \langle B, 0, 1, -, \cap, \cup \rangle$ where B is a set, 0 and 1 elements, $-$ a unary operation and $\cap$ and $\cup$ binary operations. The operations have the usual interpretation. Recall that an **atom** is an element a which sits directly above 0, that is, if $0 \leq x \leq a$ then $x = 0$ or $x = a$. In the finite case, a boolean algebra is always isomorphic to an algebra of subsets of some set X (which turns out to be the set of atoms of the algebra) with 0 interpreted as $\emptyset$, 1 interpreted as X, $-$ being the relative complement, $\cap$ and $\cup$ the intersection and union of sets. We denote this algebra by $\mathbf{2^X}$. Its carrier set is the set of functions from X to $2 = \{0, 1\}$, a typical construction of the powerset of X. A **homomorphism** of boolean algebras $\mathfrak{A}, \mathfrak{B}$ is a map $h : A \to B$ such that $h(0) = 0$, $h(1) = 1$, $h(-c) = -h(c)$, $h(c \cap d) = h(c) \cap h(d)$ and $h(c \cup d) = h(c) \cup h(d)$. From two boolean algebras, $\mathfrak{A}$ and $\mathfrak{B}$, we can form the **product**, $\mathfrak{A} \times \mathfrak{B}$, as follows

$$\mathfrak{A} \times \mathfrak{B} = \langle A \times B, \langle 0, 0 \rangle, \langle 1, 1 \rangle, -, \cap, \cup \rangle$$

where $-\langle a, b \rangle = \langle -a, -b \rangle$, $\langle a, b \rangle \cap \langle c, d \rangle = \langle a \cap c, b \cap d \rangle$ and $\langle a, b \rangle \cup \langle c, d \rangle = \langle a \cup c, b \cup d \rangle$. Important to know is that the atoms of $\mathfrak{A} \times \mathfrak{B}$ are exactly the elements $\langle a, 0 \rangle$ with a an atom of $\mathfrak{A}$ and $\langle 0, b \rangle$ with b an atom of $\mathfrak{B}$. As a consequence, if $\mathfrak{A}$ has m atoms and $\mathfrak{B}$ has n atoms, then the product has exactly $m + n$ atoms. Another construction is that of the **tensor product**. Using the representation theorem we can say that for finite algebras if $\mathfrak{A}$ is the set algebra $\mathbf{2^X}$ and $\mathfrak{B}$ the set algebra $\mathbf{2^Y}$ then the tensor product, denoted by $\mathfrak{A} \otimes \mathfrak{B}$, is isomorphic to $\mathbf{2^{X \times Y}}$; consequently, it has $m \cdot n$ atoms if $\mathfrak{A}$ has m atoms and $\mathfrak{B}$ n.

Given a set X of arbitrary propositional variables or constants the set $Tm_B(X)$ of **boolean** X**-terms** is simply the set of all terms which can be produced from X with the help of the standard connectives, such as $\top$, $\bot$, $\neg$, $\wedge$, $\vee$, etc. The X-terms modulo boolean equivalence form a boolean algebra which we denote by $\mathfrak{Fr}(X)$. Using disjunctive normal form one can show that the atoms of this algebra are exactly the (equivalence classes of) formulas

$$At_C = \bigwedge_{F \in C} F \wedge \bigwedge_{F \notin C} \neg F$$

where $C \subseteq X$. Hence, $\mathfrak{Fr}(X)$ has exactly 2^n atoms, where $n = \sharp X$.

A concept related to boolean algebras and the representation theorem is that of a *classification scheme*. We call a triple $\mathfrak{C} = \langle \mathbb{F}, \mathbb{O}, \gamma \rangle$ a **classification scheme** if $\mathbb{F}$ and $\mathbb{O}$ are sets and $\gamma \subseteq \mathbb{F} \times \mathbb{O}$ a binary relation. $\mathbb{F}$ is called the set of **features** and $\mathbb{O}$ the set of **objects**. The principal idea behind classification schemes is that we cannot access the objects by way of pointing at them with a unique label but by describing them with a boolean feature term instead. For example, whereas we have the traditional syntactic categories *n*, *np*, *c* or *cp* as primitives, we can analyse them as feature bundles by introducing the features $\mathsf{nom}, \mathsf{comp}$ and $\mathsf{lex}, \mathsf{phr}$ with obvious meaning, and the following classification table

σ	*n*	*np*	*c*	*cp*
nom	$*$	$*$		
comp			$*$	$*$
lex	$*$		$*$	
phr		$*$		$*$

This scheme will be called $\mathfrak{Cat}$. Classification schemes are *not* the same as *contexts* (for a definition of the latter see (Davey and Priestley, 1991)), although there is a strong similarity. Unlike contexts, classification schemes are imbalanced in the role of the features and the role of the objects. We associate different boolean algebras with them. Naturally, for objects we take the algebra $\mathbf{2}^{\mathbb{O}}$ of sets of objects, while for the features we take the boolean algebra $\mathfrak{Fr}(\mathbb{F})$. There are two natural maps that can be associated with γ. For each feature term t we can ask for the set of objects satisfying t; this set is the **extension** of t and is denoted by t^γ. It is computed as follows.

$$\begin{array}{lcl} \mathsf{f}^\gamma & = & \{o | \mathsf{f}\,\gamma\, o\} \\ (\neg\mathsf{t})^\gamma & = & \mathbb{O} - \mathsf{t}^\gamma \\ (\mathsf{t} \wedge \mathsf{u})^\gamma & = & \mathsf{t}^\gamma \cap \mathsf{u}^\gamma \\ (\mathsf{t} \vee \mathsf{u})^\gamma & = & \mathsf{t}^\gamma \cup \mathsf{u}^\gamma \end{array}$$

$(-)^\gamma$ will be called the **canonical map** of $\mathfrak{C}$. For example, for $\mathfrak{Cat}$ the canonical map is

$$\begin{array}{lcl} \mathsf{nom}^\sigma & = & \{\mathsf{n}, \mathsf{np}\} \\ \mathsf{comp}^\sigma & = & \{\mathsf{c}, \mathsf{cp}\} \\ \mathsf{lex}^\sigma & = & \{\mathsf{n}, \mathsf{c}\} \\ \mathsf{phr}^\sigma & = & \{\mathsf{np}, \mathsf{cp}\} \end{array}$$

Likewise, with each set Y of objects we can ask for the (equivalence class of) the most specific term containing Y. This is defined by putting $O_\gamma = \{F | F\gamma O\}$; then

$$Y_\gamma = \bigvee_{O \in Y} At_{O_\gamma}$$

Notice that $Y \subseteq (Y_\gamma)^\gamma$ but that equality need not hold. This is so because there can be objects which cannot be discriminated by features, and if Y contains only one of them, $(Y_\gamma)^\gamma$ must contain both. In the finite case (which we assume here), there is a largest number k such that there exist k different objects with identical classification. This number is denoted by $\mu(\mathfrak{C})$ and is called the **multiplicity** of the classification scheme. $\mu(\mathfrak{C}) = 1$ iff the homomorphism $(-)^\gamma : \mathfrak{Fr}(\mathbb{F}) \to 2^{\mathbb{O}}$ is surjective iff every set of objects is the extension of some $\mathbb{F}$-term (at least in the finite case). There is also a counterpart on the side of the features; if there are features whose extension can be expressed by means of other features we call $\mathfrak{C}$ **redundant**. The redundancy can be quantified by the number of features that can be eliminated in the sense that after elimination the same sets are extensions of terms. We call this number $\rho(\mathfrak{C})$. Our example has $\rho(\mathfrak{Cat}) = 2$ since $\mathsf{phr}^\sigma = -\mathsf{lex}^\sigma$ and $\mathsf{nom}^\sigma = -\mathsf{comp}^\sigma$. Notice that even if a classification scheme is nonredundant it might still be inefficient in using too many features. It can be made efficient, however, only by rearranging the primitives of the classification itself, namely the features.

Classification schemes can be extended by enlarging the feature set or the object set. The most elegant way to perform this is by combining a scheme with another. Let $\mathfrak{C} = \langle \mathbb{E}, \mathbb{O}, \gamma \rangle$ and $\mathfrak{D} = \langle \mathbb{F}, \mathbb{O}, \delta \rangle$ be classification schemes over the same objects. Then we can form the **feature addition** $\mathfrak{C}\text{ffi}\mathfrak{D}$ by taking the disjoint union $\mathbb{E} + \mathbb{F}$ of the features and the sum $\gamma + \delta$ of the relations. To be explicit, $\mathsf{g}(\gamma + \delta)o$ for $\mathsf{g} \in \mathbb{E} + \mathbb{F}$ and $o \in \mathbb{O}$ iff $\mathsf{g} \in \mathbb{E}$ and $\mathsf{g}\,\gamma\, o$ or $\mathsf{g} \in \mathbb{F}$ and $\mathsf{g}\,\delta\, o$. Then

$$\mathfrak{C}\text{ffi}\mathfrak{D} = \langle \mathbb{E} + \mathbb{F}, \mathbb{O}, \gamma + \delta \rangle$$

Suppose that $\mathbb{F}$ has k elements; then we call $\mathfrak{C}\text{ffi}\mathfrak{D}$ a k-**extension** of $\mathfrak{C}$. Now assume that $\mathfrak{C} = \langle \mathbb{F}, \mathbb{O}, \gamma \rangle$ and that $\mathfrak{D} = \langle \mathbb{F}, \mathbb{P}, \delta \rangle$. We can then add $\mathfrak{D}$ to $\mathfrak{C}$ in a similar way by taking the disjoint union of the object sets; let then $\gamma \oplus \delta$ be defined by $\mathsf{f}\,\gamma \oplus \delta\, o$ iff $o \in \mathbb{O}$ and $\mathsf{f}\,\gamma\, o$ or $o \in \mathbb{P}$ and $\mathsf{f}\,\delta\, o$.

$$\mathfrak{C} \oplus \mathfrak{D} = \langle \mathbb{F}, \mathbb{O} + \mathbb{P}, \gamma \oplus \delta \rangle$$

Alternatively, we can form pairs $\langle o, p \rangle$ with $o \in \mathbb{O}$ and $p \in \mathbb{P}$. In this case, however, it would be most natural to assume as classifying features pairs of features. If, however, we insist on having the original features, we have to eliminate all those pairs which receive different classification in $\mathfrak{C}$ and $\mathfrak{D}$. To be exact, let

$$\mathbb{O} \times_{\mathbb{F}} \mathbb{P} = \{\langle o, p \rangle | o_\gamma = p_\delta\}$$

Next, put $\mathsf{t}^{\gamma \otimes \delta} = \mathsf{t}^\gamma \times \mathsf{t}^\delta$. Then

$$\mathfrak{C} \otimes \mathfrak{D} = \langle \mathbb{F}, \mathbb{O} \times_{\mathbb{F}} \mathbb{P}, \gamma \otimes \delta \rangle$$

The notation is not accidental; the construction $\mathfrak{C} \oplus \mathfrak{D}$ yields on the object side the algebra $\mathbf{2}^{\mathbb{O}+\mathbb{P}} \cong \mathbf{2}^{\mathbb{O}} \oplus \mathbf{2}^{\mathbb{P}}$ while $\mathfrak{C} \otimes \mathfrak{D}$ yields a construction known as the

fibred tensor product. The latter is isomorphic to a quotient of the algebra $\mathbf{2}^{\mathbb{O}\times\mathbb{P}} \cong \mathbf{2}^{\mathbb{O}} \otimes \mathbf{2}^{\mathbb{P}}$. Without proof we state the following laws for multiplicites.

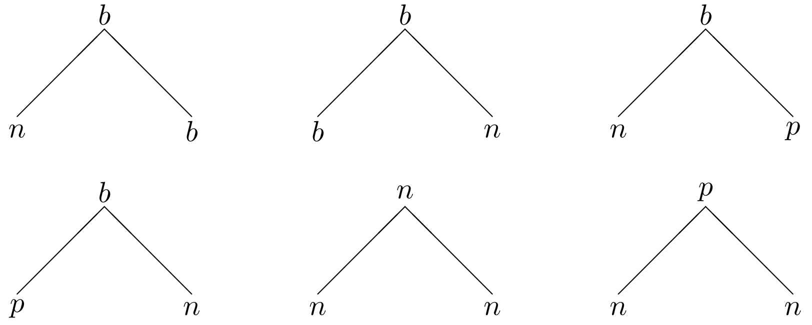
$$\begin{array}{rcl}\mu(\mathfrak{C} \oplus \mathfrak{D}) & \leq & \mu(\mathfrak{C}) + \mu(\mathfrak{D}) \\ \mu(\mathfrak{C} \otimes \mathfrak{D}) & \leq & \mu(\mathfrak{C}) \cdot \mu(\mathfrak{D})\end{array}$$

2.2. Grammars

A **context-free grammar** (cfg) is a quadruple $\mathbb{G} = \langle \mathrm{Sy}, \Sigma, \Omega, R\rangle$ where Sy is a finite set, the *set of symbols*, $\Sigma \subseteq \mathrm{Sy}$ the *set of start symbols*, $\Omega \subseteq \mathrm{Sy}$ the *set of terminal symbols* and $R \subseteq \mathrm{Sy} \times \mathrm{Sy}^*$ a finite set, the *set of rules*. Notice that this deviates from the standard definition in that there is a complete parallelism between start symbols and terminal symbols. There is a *set* of start symbols, and terminal symbols may appear on the left hand side of a rule. Typically, only a single start symbol is allowed. However, at the cost of introducing a special start symbol S together with rules $S \rightarrow T$ for each $T \in \Sigma$ we can reduce the grammar to a cfg with only a single start symbol. Secondly, the terminal symbols correspond more to preterminal symbols in standard cfgs. To get a standard cfg from cfgs as defined here one has to specify a *lexicon* in addition; a **lexicon** is a pair $\langle L, c\rangle$, where L is a finite set and $c : L \rightarrow \Omega$ a map. c is also called the *syntactic classification map* because it tells us what syntactic category a word or lexical entry belongs to. An (ordered) tree is an object $\mathfrak{T} = \langle T, r, <, \rangle$, where T is a finite set and $< \subseteq T^2$ a tree order with r being the root, and a precedence ordering. If one thinks of the nodes as representing events (e. g. the event of uttering the constituent denoted by that node), $<$ then is the ordering of proper subevents and that of (strict) precendence. A **labelled tree** with labels in Sy is a pair $\langle \mathfrak{T}, \ell\rangle$ where $\ell : T \rightarrow \mathrm{Sy}$ returns a label from Sy for each node from $\mathfrak{T}$. A cfg generates $\langle \mathfrak{T}, \ell\rangle$ just in case (1) the root has a label from Σ, (2) the leaves have labels from Ω and (3) if x immediately dominates $y_1, \ldots, y_n$ (in that order) then $\ell(x) \rightarrow \ell(y_1) \ldots \ell(y_n) \in R$. Clause (3) is called the **local admissibility condition**.

Context free grammars form the underlying machine of *feature grammars*. A **feature grammar** is a pair $\mathfrak{G} = \langle \mathbb{G}, \mathfrak{C}\rangle$ where $\mathbb{G}$ is a cf-grammar over the set Sy of symbols and $\mathfrak{C}$ a classification scheme over Sy. Given that $\mathfrak{C} = \langle \mathbb{F}, \mathrm{Sy}, \gamma\rangle$ we also say that $\mathfrak{G}$ is an $\mathbb{F}$**-grammar**. The structures generated by feature grammars are not simply labelled trees but **feature trees**. Feature tree are pairs $\langle \mathfrak{T}, \phi\rangle$ where $\phi : T \rightarrow \mathfrak{Fr}(\mathbb{F})$; thus, each node is assigned an $\mathbb{F}$-term. In some sense we use the cf-grammar as a machine to distribute the features over the trees. Throughout this paper it is feature-grammars we are dealing with as opposed to ordinary cfgs. This might seem to be an unnecessary complication, but the additional twist is that when the classification scheme has multiplicity > 1 there are symbols in Sy which are undifferentiable with the help of the features. They will play a central role in the theory of codes. To give an example, suppose that we have a single feature f;

it is possible to write an $\{\mathsf{f}\}$-grammar which distributes f on exactly one node in the tree. For example this binary branching grammar.



Here $\Sigma = \{p, b\}$ and $\Omega = \{p, n\}$. Put $\mathsf{f}^{\sigma} = \{p\}$. Then first of all, f can appear only at the root or as a daughter of a b-node. The b-nodes on the other hand are filed along a single branch until a p-node is reached. Since a p-node only has n-daughters, no p-node can dominate another p-node. Together this shows that there is at most one p-node. With $n \notin \Sigma$ we make sure that any tree we start can be completed to contain a p-node and by $b \notin \Omega$ we make sure that it must be completed in this way. Thus exactly one p-node appears, which had to be proved. It is not hard to show that two symbols cannot guarantee the correct distribution of f. Since this is the case, no grammar exists generating the required trees in which the symbols can be differentiated by means of the distributed symbol f alone. We say in this case that the grammar needs a *memory.* To be precise, let us state the following definitions.

An n-extension of an $\mathbb{F}$-grammar $\langle \mathbb{G}, \mathfrak{C} \rangle$ is a $\mathbb{F} \cup \mathbb{E}$-grammar $\langle \mathbb{G}, \mathfrak{C}\text{ffi}\mathfrak{D} \rangle$ where $\mathfrak{D}$ is a classification scheme over $\mathbb{E} = \{\mathsf{e}_1, \ldots, \mathsf{e}_n\}$. So, an n-extension differs only with respect to the features; the underlying cfg is completely identical. The extending grammar has an additional resource of n features which might be used to discriminate more symbols than in the original grammar. A grammar can always be extended, but this only makes sense if the canonical map of the classification scheme, $(-)^{\gamma}$, is not surjective. If that is the case, it is clear that there is a finite number of extra features that need be added to make the feature map surjective.

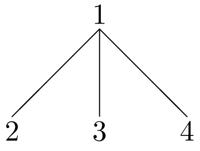
Definition 2.1. A feature-grammar $\langle \mathbb{G}, \mathfrak{C} \rangle$ is called ***rational*** if the map $(-)^{\gamma} : \mathfrak{Fr}(\mathbb{F}) \to 2^{\mathrm{Sy}}$ is surjective. The smallest number k such that there exists a rational k-extension is called the ***memory*** of a feature-grammar. Thus, a feature-grammar is rational iff it has memory 0.

There are explicit ways to calculate the memory. Namely, the memory is $\Gamma 0E70^2 \log\mu(\mathfrak{C})\urcorner$ where $\mu(\mathfrak{C})$ is the multiplicity of the classification scheme and $\Gamma 0E70k\urcorner$ denotes the smallest integer $\geq k$. This follows from the fact that r features allow to discriminate 2^r symbols. Explicit examples of memory can be found in connection with long-distance dependencies. In GPSG the device of the slash-feature takes care of the distribution of empty categories; in GB, this slash-feature is not explicit in the trees. Feature-grammars offer a way to uphold the GB-analysis while retaining a cf-grammar underneath.

Finally, let us remark that if a feature grammar is rational the underlying cfg can in effect be presented via the feature system, since every element of Sy can be singled out by a feature term. This is the way we will present grammars here in sequel, to save space (and mental energy). The memory will be instatiated via auxiliary features. Analogously, we can pretend that our grammar produces feature trees directly. This requires adapting the definition of generation. The grammar generates a feature tree if (1$'$) the root has label a and $\mathsf{a} \leq \mathsf{s}$, where s is the start term, (2$'$) the terminal nodes have label b with $\mathsf{b} \leq \mathsf{o}$, o the stop term, (3$'$) If x dominates $y_1 \ldots y_m$ then there is a rule $\mathsf{a} \to \mathsf{b}_1 \ldots \mathsf{b}_m$ such that $\ell(x) \leq \mathsf{a}$ and $\ell(y_j) \leq \mathsf{b}_j$ for all $j \leq m$.

3. Grammars and Logic

There is a special language for describing facts about trees which is perfectly fitted to the study of codes. This language is a certain fragment of dynamic logic. Chiefly, this language allows to introduce a battery of binary relations over trees which can be used to express both local and nonlocal dependencies between nodes of the trees. The fundamental relations with which we start are the relations in a *local tree*. They are in particular $\Diamond_{\uparrow}$ *daughter of*, $\Diamond_{\downarrow}$ *mother of*, $\Diamond_{\rightarrow}$ *immediate right sister of* and $\Diamond_{\leftarrow}$ *immediate left sister of*.



$$\begin{array}{rcl} \Diamond_{\uparrow} & = & \{\langle 2,1\rangle, \langle 3,1\rangle, \langle 4,1\rangle\} \\ \Diamond_{\downarrow} & = & \{\langle 1,2\rangle, \langle 1,3\rangle, \langle 1,4\rangle\} \\ \Diamond_{\rightarrow} & = & \{\langle 2,3\rangle, \langle 3,4\rangle\} \\ \Diamond_{\leftarrow} & = & \{\langle 4,3\rangle, \langle 3,2\rangle\} \end{array}$$

From these relations we can form complex relations with the help of the set-theoretic *union* $\cup$, the *composition* $\circ$ and the *Kleene Star* *. The reader may check the correctness of the following list of relations. (Recall that R^+ is defined by $R \circ R^*$.)

Relation	Translation
$\Diamond_{\downarrow}^{*}$	*dominates*
$\Diamond_{\downarrow}^{+}$	*properly dominates*
$\Diamond_{\uparrow}^{*}$	*is dominated by*
$\Diamond_{\uparrow}^{+}$	*is properly dominated by*
$\Diamond_{\leftarrow}^{+} \cup \Diamond_{\rightarrow}^{+}$	*sister of*
$\Diamond_{\uparrow}^{*} \circ \Diamond_{\rightarrow}^{+} \circ \Diamond_{\downarrow}^{*}$	*precedes*
$\Diamond_{\uparrow}^{*} \circ \Diamond_{\leftarrow}^{+} \circ \Diamond_{\downarrow}^{*}$	*succeeds*
$\Diamond_{\uparrow}^{*} \cup \Diamond_{\downarrow}^{*}$	*overlaps with*
$\Diamond_{\uparrow}^{*} \circ (\Diamond_{\leftarrow}^{+} \cup \Diamond_{\rightarrow}^{+}) \circ \Diamond_{\downarrow}^{*} \cup \Diamond_{\uparrow}^{+} \cup \Diamond_{\downarrow}^{+}$	*is different from*

Suppose now that we want to describe $\mathbb{F}$-trees. Then first of all we take a constant f_i for each $F_i \in \mathbb{F}$ in addition to the usual variables $p_1, p_2, \ldots$; we grant ourselves the usual boolean connectives $\neg, \wedge, \vee, \rightarrow, \ldots$. And, finally, we allow a relation R and a proposition ϕ to combine to the proposition $R\phi$ which is true at a node x exactly if there exists a y standing in relation R to x which satisfies ϕ. [1] There is at last another connective, namely ?, the so-called *test*, which applies to propositions and returns a relation. We have $\langle x, y \rangle \in \phi?$ iff $x = y$ and ϕ holds at x. The collection of all propositions which can be formed in this way using the fundamental relations is called the **orientation language** and denoted by $\mathit{Olt}(\mathbb{F})$. Its expressive power is quite large, allowing to state non-local dependencies along regular paths, and in fact anything that typically is required in linguistic theories. In this language one can also capture finite $\mathbb{F}$-trees axiomatically; the resulting logic will be called Φ. The precise content of the axioms is of no importance here. The reader is referred to (Harel, 1984) for details. In the sequel, the following theorems of dynamic logic will be made use of.

$$\begin{array}{llcl}
(d\circ) & R \circ S\,\phi & .\leftrightarrow. & R\,(S\,\phi) \\
(d?) & \phi?\,\psi & .\leftrightarrow. & \phi \wedge \psi \\
(d\cup) & (R \cup S)\phi & .\leftrightarrow. & R\,\phi \vee S\,\phi \\
(d\,^*) & R^*\,\phi & .\leftrightarrow. & \phi \vee R \circ R^*\,\phi
\end{array}$$

[1] We confuse pograms and modal operators here. In dynamic logic one has to write $\langle R \rangle \phi$ instead of just $R\phi$.

Another useful operator, usually not present in dynamic logic, is the *relational converse*. We define $R = \{\langle y, x\rangle | \langle x, y\rangle \in R\}$. The following then holds.

$$\begin{array}{lll}
(cc) & (R) & = R \\
(c\Diamond_{\uparrow}) & \Diamond_{\uparrow} & = \Diamond_{\downarrow} \\
(c\Diamond_{\leftarrow}) & \Diamond_{\leftarrow} & = \Diamond_{\rightarrow} \\
(c\circ) & (R \circ S) & = S \circ R \\
(c?) & (\phi?) & = \phi? \\
(c\cup) & (R \cup S) & = R \cup S \\
(c\,^*) & (R^*) & = (R)^*
\end{array}$$

These postulates show that if the converse operator is added it does not increase the power of the orientation language.

A rational $\mathbb{F}$-grammar can be transformed into an axiom in the orientation language as follows. First, note that by rationality each rule $\rho = u \to v_1 \ldots v_m$ can be rewritten as $\mathsf{u} \to \mathsf{v}_1 \ldots \mathsf{v}_m$, where u, v_i are suitable boolean $\mathbb{F}$-terms such that $\mathsf{u}^\gamma = \{u\}$ and $\mathsf{v}_i^\gamma = \{v_i\}$ for $i \leq m$, $(-)^\gamma$ the canonical map induced by the classification scheme of $\mathbb{G}$. Now put

$$\lambda(\rho) = \mathsf{u} \wedge \Diamond_{\downarrow}(\neg\Diamond_{\leftarrow}\top \wedge \mathsf{v}_1 \wedge \Diamond_{\rightarrow}(\mathsf{v}_2 \wedge \Diamond_{\rightarrow}(\ldots \wedge \Diamond_{\rightarrow}(\mathsf{v}_m \wedge \neg\Diamond_{\rightarrow}\top)\ldots)))$$

There exists a s such that $\mathsf{s}^\sigma = \Sigma$ and a o such that $\mathsf{o}^\sigma = \Omega$. We can express the boundary conditions on the trees by

$$\begin{array}{lll}
\lambda_\Sigma & = & \neg\Diamond_{\uparrow}^*(\neg\Diamond_{\uparrow}\top \wedge \neg\mathsf{s}) \\
\lambda_\Omega & = & \neg\Diamond_{\downarrow}^*(\neg\Diamond_{\downarrow}\top \wedge \neg\mathsf{o})
\end{array}$$

Finally, for $\mathfrak{G} = \langle \mathbb{G}, \sigma\rangle$ let

$$\lambda(\mathfrak{G}) = \lambda_\Sigma \wedge \lambda_\Omega \wedge (\neg\mathsf{o} \to \bigvee_{\rho \in R} \lambda(\rho))$$

The local admissibility condition is captured by the third conjunct, as is readily checked. In this way, a rational $\mathbb{F}$-grammar corresponds to a single axiom extending Φ, which is variable free. We call $\lambda(\mathfrak{G})$ the **characteristic axiom** of $\mathfrak{G}$.

4. Grammar Manipulations

If we take the analogy of grammars and axiomatic descriptions of trees literally, there should be constructions on grammars which mirror the usual logical connectives, in particular conjunction and disjunction. Indeed, there are such constructions and they will turn out to be central tools in *grammar refinement.* To begin, let us introduce certain constructions for context-free grammars. Given a single cfg $\mathbb{G} = \langle \mathrm{Sy}, \Sigma, \Omega, R\rangle$ and a set $C \subseteq \mathrm{Sy}$ we write

$$\mathbb{G} \restriction C = \langle C, \Sigma \cap C, \Omega \cap C, R \cap C \times C^*\rangle$$

and call $\mathbb{G} \upharpoonright C$ the **restriction of** $\mathbb{G}$ **to** C. We also say that $\mathbb{G} \upharpoonright C$ is obtained from $\mathbb{G}$ by *cutting* $\mathrm{Sy} - C$, the complement of C in Sy. The intended effect is that we remove all symbols from the grammar which do not belong to C and all those rules which employ these symbols.

Given two cfgs, $\mathbb{G}_1 = \langle \mathrm{Sy}_1, \Sigma_1, \Omega_1, R_1\rangle$ and $\mathbb{G}_2 = \langle \mathrm{Sy}_2, \Sigma_2, \Omega_2, R_2\rangle$ let us define the grammar $\mathbb{G}_1 \times \mathbb{G}_2$. It operates on the set $\mathrm{Sy}_1 \times \mathrm{Sy}_2$, thus pairs of symbols, one from $\mathbb{G}_1$ and the other from $\mathbb{G}_2$. The start symbols are simply those of $\Sigma_1 \times \Sigma_2$, the terminal symbols those from $\Omega_1 \times \Omega_2$ and the rules are a straightforward marriage of the respective local conditions:

$$(*) \qquad \begin{array}{l} \langle u^1, u^2\rangle \to \langle v^1_1, v^2_1\rangle \ldots \langle v^1_m, v^2_m\rangle \in R_1 \times R_2 \text{ iff} \\ u_1 \to v^1_1 \ldots v^1_m \in R_1 \text{ and } u_2 \to v^2_1 \ldots v^2_m \in R_2 \end{array}$$

The trees generated by this grammar are of the form $\langle \mathfrak{T}, \ell_1 \times \ell_2\rangle$, where $\ell_i : T \to \mathrm{Sy}_i$.

Proposition 4.1. $\langle \mathfrak{T}, \ell_1 \times \ell_2\rangle$ is generated by $\mathbb{G}_1 \times \mathbb{G}_2$ iff $\langle \mathfrak{T}_1, \ell_1\rangle$ is generated by $\mathbb{G}_1$ and $\langle \mathfrak{T}, \ell_2\rangle$ is generated by $\mathbb{G}_2$. $\dashv$

The proof is by straightforward checking of the definitions. Notice, for example, that $(*)$ says exactly that the local admissibility condition for $\mathbb{G}_1 \times \mathbb{G}_2$ is the conjunction of the local admissibility conditions for $\mathbb{G}_1$ for the left factor and for $\mathbb{G}_2$ for the right factor. Now we lift this construction to feature-grammars. Given two $\mathbb{F}$-grammars $\mathfrak{G}_1 = \langle \mathbb{G}_1, \mathfrak{C}_1\rangle$ and $\mathfrak{G}_2 = \langle \mathbb{G}_2, \mathfrak{C}_2\rangle$ we want as a result an $\mathbb{F}$-grammar. Thus we put

$$\mathfrak{G}_1 \otimes \mathfrak{G}_2 = \langle \mathbb{G}_1 \times \mathbb{G}_2 \upharpoonright (\mathrm{Sy}_1 \times_{\mathbb{F}} \mathrm{Sy}), \mathfrak{C}_1 \otimes \mathfrak{C}_2\rangle$$

Proposition 4.2. Let $\mathfrak{G}_1, \mathfrak{G}_2$ be $\mathbb{F}$-grammars. Then the memory of $\mathfrak{G}_1 \otimes \mathfrak{G}_2$ is at most the sum of memories of $\mathfrak{G}_1$ and $\mathfrak{G}_2$. In particular, $\mathfrak{G}_1 \otimes \mathfrak{G}_2$ is memory free if both factors are. Let $\mathfrak{T}$ be an $\mathbb{F}$-tree. Then $\mathfrak{G}_1 \otimes \mathfrak{G}_2$ generates $\mathfrak{T}$ iff both $\mathfrak{G}_1$ and $\mathfrak{G}_2$ generate $\mathfrak{T}$. $\dashv$

Similarly, a grammar $\mathbb{G}_1 + \mathbb{G}_2$ can be defined which uses as symbols the set $\mathrm{Sy}_1 + \mathrm{Sy}_2$ and is defined by

$$\mathbb{G}_1 + \mathbb{G}_2 = \langle \mathrm{Sy}_1 + \mathrm{Sy}_2, \Sigma_1 + \Sigma_2, \Omega_1 + \Omega_2, R_1 + R_2\rangle$$

Proposition 4.3. Suppose that $\mathbb{G}_1$ and $\mathbb{G}_2$ are grammars over disjoint sets of symbols. Then $\mathbb{G}_1 + \mathbb{G}_2$ generates $\mathfrak{T}$ iff either $\mathbb{G}_1$ generates $\mathfrak{T}$ or $\mathbb{G}_2$ does. $\dashv$

Adding the classifying component is as straightforward. We put

$$\mathfrak{G}_1 \oplus \mathfrak{G}_2 = \langle \mathbb{G}_1 + \mathbb{G}_2, \mathfrak{C}_1 \oplus \mathfrak{C}_2\rangle$$

Proposition 4.4. Let $\mathfrak{G}_1, \mathfrak{G}_2$ be $\mathbb{F}$-grammars. Then the memory of $\mathfrak{G}_1 \oplus \mathfrak{G}_2$ is at most 1 plus the sum of memories of $\mathfrak{G}_1$ and $\mathfrak{G}_2$. Let $\mathfrak{T}$ be an $\mathbb{F}$-tree. Then $\mathfrak{G}_1 \oplus \mathfrak{G}_2$ generates $\mathfrak{T}$ iff either of $\mathfrak{G}_1$ and $\mathfrak{G}_2$ generates $\mathfrak{T}$. $\dashv$

Notice that as regards the size of the memory we might need an additional feature that tells apart the symbols from $\mathfrak{G}_1$ from those of $\mathfrak{G}_2$.

5. The Coding Theorem

Standard syntactical theories proceed in the definition of languages or grammars in the following way. They first specify a rather rudimentary grammar, for example $\overline{X}$-syntax and then add a set of principles or conditions, which further restrict the possible trees. This strategy can be mirrored by the method of *grammar refinement.* We start with the same rudimentary grammar $\mathfrak{X} = \langle \mathbb{X}, \mathfrak{C} \rangle$ and take the first principle, P_1. Assuming that P_1 uses no features unknown to $\mathfrak{X}$, which can always be achieved by passing from $\mathfrak{X}$ to a suitable extension, we then try to write a grammar which produces as output exactly the trees conforming to P_1. If we succeed, call this grammar $\mathfrak{P}_1$. Now form $\mathfrak{X} \otimes \mathfrak{P}_1$; this is called the *refinement of* $\mathfrak{X}$ by P_1. Proceed with P_2. And so on. Having succeeded for all principles we end up with a grammar which is correct for $\mathfrak{X}$ and all principles. The success of this strategy rests on the possibility to define such grammars $\mathfrak{P}_i$. Thus we are interested in the question of which principles allow to be converted into grammars. It is the case, however, that the possibility of doing this varies with the grammar into which we want to code the principle.

Definition 5.1. Let P be a condition on $\mathbb{F}$-trees and let $\mathfrak{G}$ be an $\mathbb{F}$-grammar. We say that $\mathfrak{G}$ ***codes*** P if $\mathfrak{G}$ generates $\mathfrak{T}$ exactly if $\mathfrak{T}$ satisfies P.

The problem of codability will be attacked in a slightly different way. We begin with the set $\mathbb{F}$ and take a disjoint set $\mathbb{X} = \{\mathsf{x}_1, \dots, \mathsf{x}_m\}$. An $\mathbb{X}$-**spread** over $\mathbb{F}$-trees is a condition on how to distribute the features from $\mathbb{X}$ over the $\mathbb{F}$-trees. Alternatively, a spread is a set of $\mathbb{X}$-extensions of $\mathbb{F}$-trees. A spread is *Olt*-**definable** if there exist $\phi_1, \dots, \phi_m \in \mathit{Olt}(\mathbb{F})$ such that

$$(\dagger) \qquad \begin{array}{rcl} \mathsf{x}_1 & \leftrightarrow & \phi_1(\mathsf{x}_1, \dots, \mathsf{x}_m) \\ \mathsf{x}_2 & \leftrightarrow & \phi_2(\mathsf{x}_2, \dots, \mathsf{x}_m) \\ & & \dots \\ \mathsf{x}_m & \leftrightarrow & \phi_m(\mathsf{x}_1, \dots, \mathsf{x}_m) \end{array}$$

Even though such a definition can be made for all trees, we cannot write a simple grammar coding this spread into *all* grammars in the sense discussed above. Nevertheless, if we restrict ourselves to at most k-branching trees there will be such a grammar as asserts the next theorem. Moreover, if $\mathfrak{C}_k$ codes such a spread into at most k-branching trees and $\mathfrak{C}_\ell$ codes it into at most ℓ-branching trees with $\ell \leq k$

then $\mathfrak{C}_\ell$ results from $\mathfrak{C}_k$ by removing all rules which produce more than ℓ daughters. This is easy to see. Thus, let us restrict now the attention to at most k-branching trees, k a natural number. For the problem mentioned at the beginning this is naturally satisfied because starting with a grammar means restricting the set of trees to trees with bounded branching number.

Theorem 5.2 (Coding Theorem). Let k be a natural number. Any $Olt(\mathbb{F})$-definable spread over at most k-branching trees is codable.

The proof proceeds by simplifying the definitional schema for the spread by introducing additional variables. Let us attack, for example, the first equivalence of the schema. Assume ϕ_1 is a boolean function f of the propositions $\psi_1(\bar{\mathsf{x}}), \dots, \psi_k(\bar{\mathsf{x}})$. (For example, $\phi_1 = \neg(\psi_1 \vee (\psi_2 \wedge \psi_3))$.) Introduce new variables $\mathsf{y}_1, \dots, \mathsf{y}_k$ and rewrite the first equivalence into

$$(\ddagger) \qquad \begin{array}{lcl} \mathsf{x}_1 & \leftrightarrow & f(\mathsf{y}_1, \dots, \mathsf{y}_k) \\ \mathsf{y}_1 & \leftrightarrow & \psi_1(\mathsf{x}_1, \dots, \mathsf{x}_m) \\ \mathsf{y}_2 & \leftrightarrow & \psi_2(\mathsf{x}_1, \dots, \mathsf{x}_m) \\ & & \dots \\ \mathsf{y}_k & \leftrightarrow & \psi_k(\mathsf{x}_1, \dots, \mathsf{x}_m) \end{array}$$

If we replace the first line in (†) by the system (‡) then we obtain a system in which the complexity of the formula ϕ_1 is reduced.

Suppose next that $\phi = (R \circ S)\psi$. Using $(d\circ)$ we can rewrite this into

$$\begin{array}{lcl} \mathsf{x}_1 & \leftrightarrow & R\mathsf{y} \\ \mathsf{y} & \leftrightarrow & S\psi(\mathsf{x}_1, \dots, \mathsf{x}_m) \end{array}$$

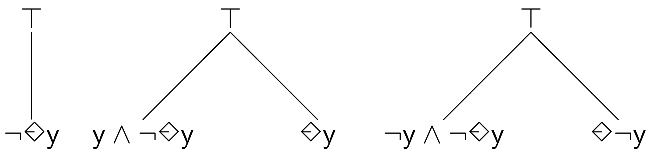
Similarly with $\phi = (R \cup S)\psi$, which is equivalent to $R\psi \vee S\psi$, by $(d\cup)$. And with $\phi = (\psi?)\chi$ which is equivalent to $\psi \wedge \chi$ by $(d?)$. Finally, let $\phi = R^*\,\psi$. By (d^*), $R^*\,\psi$ is equivalent to $\psi \vee R \circ R^*\,\psi$. Thus we can rewrite $\mathsf{x}_1. \leftrightarrow .R^*\psi(\bar{\mathsf{x}})$ into

$$\begin{array}{lcl} \mathsf{x}_1 & \leftrightarrow & \mathsf{y} \vee R\mathsf{x}_1 \\ \mathsf{y} & \leftrightarrow & \psi(\mathsf{x}_1, \dots, \mathsf{x}_m) \end{array}$$

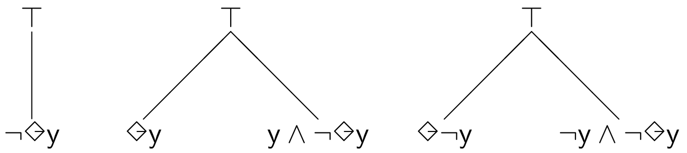
We are now down to a few basic cases; we need to show that the basic relations can be coded and that booleans can be coded, and we are done. The booleans, however, are codable; consider, namely, the spread $\mathsf{x}_1 \leftrightarrow f(\mathsf{x}_1, \dots, \mathsf{x}_m)$ where f is boolean function. Then this spread is coded simply by restricting the grammar to the set of nodes satisfying $\mathsf{x}_1 \leftrightarrow f(\mathsf{x}_1, \dots, \mathsf{x}_m)$. Moreover, notice that it does not matter whether x_1 appears to the right. So, the definition of the spread can be downright circular but the argument works nevertheless.

Finally, consider the basic relations ⬖, ⬗, ⬘ und ⬙. Below we list the grammars coding the spreads $\mathsf{x}. \leftrightarrow .R\mathsf{y}$ for all four relations. For the sake of simplicity we assume a grammar which is at most binary branching. Furthermore, for ease of understanding we write $R\mathsf{y}$ rather than x.

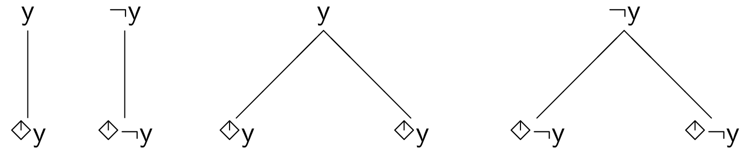
Left. $\Sigma = \neg\overset{\leftarrow}{\Diamond}\mathsf{y}$, $\Omega = \top$. The rules are


$\top$
$\top$
$\top$
$\neg\overset{\leftarrow}{\Diamond}\mathsf{y}$
$\mathsf{y} \wedge \neg\overset{\leftarrow}{\Diamond}\mathsf{y}$
$\overset{\leftarrow}{\Diamond}\mathsf{y}$
$\neg\mathsf{y} \wedge \neg\overset{\leftarrow}{\Diamond}\mathsf{y}$
$\overset{\leftarrow}{\Diamond}\neg\mathsf{y}$

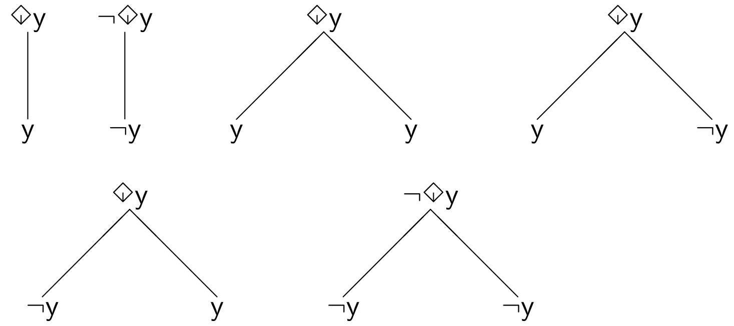
Right. $\Sigma = \neg\overset{\rightarrow}{\Diamond}\mathsf{y}$, $\Omega = \top$. The rules are


$\top$
$\top$
$\top$
$\neg\overset{\rightarrow}{\Diamond}\mathsf{y}$
$\overset{\rightarrow}{\Diamond}\mathsf{y}$
$\mathsf{y} \wedge \neg\overset{\rightarrow}{\Diamond}\mathsf{y}$
$\overset{\rightarrow}{\Diamond}\neg\mathsf{y}$
$\neg\mathsf{y} \wedge \neg\overset{\rightarrow}{\Diamond}\mathsf{y}$

Above. $\Sigma = \neg\overset{\uparrow}{\Diamond}\mathsf{y}$, $\Omega = \top$. The rules are


y
$\neg\mathsf{y}$
y
$\neg\mathsf{y}$
$\overset{\uparrow}{\Diamond}\mathsf{y}$
$\overset{\uparrow}{\Diamond}\neg\mathsf{y}$
$\overset{\uparrow}{\Diamond}\mathsf{y}$
$\overset{\uparrow}{\Diamond}\mathsf{y}$
$\overset{\uparrow}{\Diamond}\neg\mathsf{y}$
$\overset{\uparrow}{\Diamond}\neg\mathsf{y}$

Below. $\Sigma = \top$, $\Omega = \neg\overset{\downarrow}{\Diamond}\mathsf{y}$. The rules are


$\overset{\downarrow}{\Diamond}\mathsf{y}$
$\neg\overset{\downarrow}{\Diamond}\mathsf{y}$
$\overset{\downarrow}{\Diamond}\mathsf{y}$
$\overset{\downarrow}{\Diamond}\mathsf{y}$
y
$\neg\mathsf{y}$
y
y
y
$\neg\mathsf{y}$
$\overset{\downarrow}{\Diamond}\mathsf{y}$
$\neg\overset{\downarrow}{\Diamond}\mathsf{y}$
$\neg\mathsf{y}$
y
$\neg\mathsf{y}$
$\neg\mathsf{y}$

The proof of the theorem is now complete. $\dashv$

Let us spell out some consequences of the Coding Theorem. Recall that a cf-grammar is called **linear** if all rules are of the form $A \to w$ with $A \notin \Omega$ and w containing at most one occurrence of a symbol not from Ω; and that a cf-grammar is **right regular** (**left regular**) if all rules are of the form $A \to wB$ ($A \to Bw$) with $A \notin \Omega$ and $w \in \Omega^*$. A grammar is **regular** if it is either left-regular or right-regular. Suppose now that $\mathbb{G}$ is linear (left-regular, right-regular); then $\mathbb{G} \otimes \mathbb{H}$ is linear (left-regular, right-regular) as well. Let $\mathfrak{G}$ be an $\mathbb{F}$-grammar and ϕ be a $Olt(\mathbb{F})$-expressible condition on $\mathfrak{G}$. Let us now denote by $\mathfrak{G} \downarrow \phi$ the result of coding ϕ into $\mathfrak{G}$, that is, of writing a grammar that satisfies exactly the condition ϕ and the condition of $\mathfrak{G}$.

Corollary 5.3. Let $\mathfrak{G}$ be a linear (left-regular, right-regular) $\mathbb{F}$-grammar and ϕ a constant $\mathit{Olt}(\mathbb{F})$-term. Then $\mathfrak{G} \downarrow \phi$ is linear (left-regular, right-regular) as well. $\dashv$

For a proof we simply note that $\mathfrak{G} \downarrow \phi$ is isomorphic to $\mathfrak{G} \otimes \mathfrak{H}$ for an $\mathbb{F}$-grammar $\mathfrak{H}$.

Moreover, let k be a natural number. Let us denote the set of all rational $\mathbb{F}$-grammars with branching number at most k by $\mathit{Gram}_k(\mathbb{F})$. There is a least specific grammar, 1_k, with $\Sigma = \top, \Omega = \top$ and all possible rules (there are only finitely many). There is a most specific grammar, with no rules at all. Call this grammar 0_k. On $\mathit{Gram}_k(\mathbb{F})$ we can define a *conjunction* by $\otimes$, a disjunction by $\oplus$ and even a *negation.* Namely, let $\lambda(\mathfrak{G})$ be the characteristic axiom of $\mathfrak{G}$. It is a constant $\mathit{Olt}(\mathbb{F})$-term; so is the negation $\neg\lambda(\mathfrak{G})$. Hence the latter is codable into 1_k and yields a grammar $\ominus\mathfrak{G}$. Let us furthermore observe that

$$\begin{array}{lcl} \lambda(\mathfrak{G} \oplus \mathfrak{H}) & .\leftrightarrow. & \lambda(\mathfrak{G}) \vee \lambda(\mathfrak{H}) \\ \lambda(\mathfrak{G} \otimes \mathfrak{H}) & .\leftrightarrow. & \lambda(\mathfrak{G}) \wedge \lambda(\mathfrak{H}) \\ \lambda(\ominus\mathfrak{G}) & .\leftrightarrow. & \neg\lambda(\mathfrak{G}) \end{array}$$

Define now the **grammar space** as follows

$$\mathfrak{Gram}_k(\mathbb{F}) = \langle \mathit{Gram}_k(\mathbb{F}), 1_k, 0_k, \ominus, \otimes, \oplus \rangle$$

Theorem 5.4. $\mathfrak{Gram}_k(\mathbb{F})$ is a boolean algebra. $\dashv$

Let us return to the example of a feature grammar distributing a feature at a single point. There we can also write down the following formula

$$\mathsf{f}. \leftrightarrow .\neg\langle\uparrow\rangle^* \circ (\langle\leftarrow\rangle^+ \cup \langle\rightarrow\rangle^+) \circ \langle\downarrow\rangle^* \cup \langle\leftarrow\rangle^+ \cup \langle\rightarrow\rangle^+\mathsf{f}$$

A tree satisfies this formula iff f is true at a single node. Despite the fact that there is no memory free grammar coding this spread, there is a formula containing only f and no auxiliary symbols. It is tempting to believe that it is always possible to write down a charactistic formula for a feature grammar only employing the features of that grammar. However, this conclusion is premature. We sketch a counterexample. Consider a grammar generating ternary branching trees that distributes a symbol x at the leaves of a binary branching subtree and f at the leaves of that subtree. Now remove the feature x, creating a grammar with memory. There is no formula in the constant f that can distribute f exactly at the leaves of a binary branching subtree.

6. Applications of the Coding Theorem

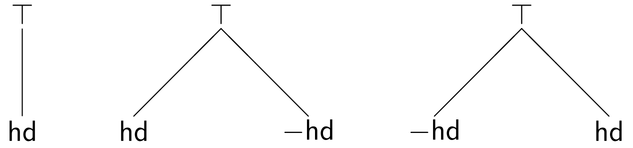
GPSG AS A FEATURE GRAMMAR. From the perspective of the Coding Theorem we can understand GPSG as the result of coding certain principles for local and non-local dependencies. In this chapter we will analyse how one can make sense of GPSG in this way and gain a deeper understanding of the relationship between GPSG and grammars based on principles and parameters. We begin by giving a sketch on how GPSG can be reduced to a feature grammar. Given the actual shape of GPSG (which assume to be that of (Gazdar *et al.*, 1985)) this is almost straightforward.

The logical engine of GPSG allows both so called atomic-valued features and features which take a feature-complex – or, as we prefer to say, a *feature term* – as a value. The former are called **type 0** features, the latter **type 1** features. Furthermore, many type 0 features are quite simple in that they allow an a priori fixed number of alternatives, while the others, such as (agr and slash), are slightly less straightforward. They are, however, not iterable and so in effect there are only a finite number of non-equivalent terms – though this number is quite large. This means that although syntactically GPSG has a language which is richer than that of boolean logic, it can be reduced to the latter. It should be noted, though, that the feature language allows a more concise definition of categories and rules. After categories are defined, GPSG names a number of *constraints*, which delimit the space of categories. These constraints are simply axioms – boolean axioms after reduction. Moreover, there exist certain *defaults* on feature specification; in our context there is no room for defaults – they have to be unravelled into ordinary boolean conditions.

The next step is the definition of the *rules*. There are two kinds of rules. The first kind are the usual rules; the second type, the so-called *metarules* state that if a rule of certain type is admitted so is a rule of another type. Metarules have the flavour of transformations; however, while transformations are part of the syntactic derivation itself, metarules are not. Metarules do not simply correspond to axioms since they produce rules, and rules are used in disjunction while axioms are used in conjunction. But GPSG allows to unravel rules plus metarules into a set of rules since the resulting set is finite.

X-BAR SYNTAX. The short rundown of GPSG illustrates that it fits rather well into the schema of feature grammars. More illuminating than this is the converse stategy of taking a GB-like grammar and produce a GPSG-style equivalent grammar. To start we show how to produce $\overline{X}$-syntax. To begin, we need four different features, hd (*head*), sp (*specifier*), co (*complement*) and ad (*adjunct*). Now the following grammar can be defined.

Centricity Grammar. $\Sigma = \mathsf{hd}$, $\Omega = \mathsf{hd}$.



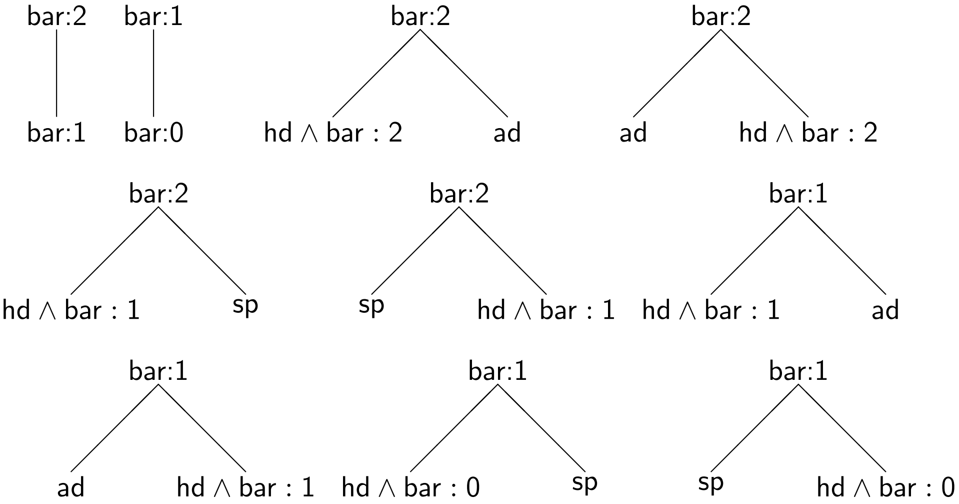
The centricity grammar does nothing but distribute the roles of head, specifier, complement and adjunct. The latter three cannot really be distinguished in terms of rules but will play their respective role later on. There is a list of features, called *head features*, which obey the following postulates, called *head feature conventions.*

HFC I. For all head-features f: $\mathsf{f} \wedge \Diamond_{\downarrow}\top. \rightarrow .\Diamond_{\downarrow}(\mathsf{hd} \wedge \mathsf{f})$
HFC II. For all head-features f: $\mathsf{hd} \wedge \mathsf{f} \wedge \Diamond_{\uparrow}\top. \rightarrow .\Diamond_{\uparrow}\mathsf{f}$

These conventions say that if a head feature term is true of the mother then it is true the head daughter and conversely. (Here, instead of head features we must speak of a *head feature term*, which is a feature term over the head features.) The HFCs stated here are pretty much those implicit in GB but different from those in GPSG, which are more complex in character. Also, unlike GPSG the notion of a *head* is not metagrammatical but inbuilt. It is rather delicate to say exactly what the head is when given a local tree, so one is advised to take the head as a primitive.

Next we introduce the *levels*. There are three levels, $\mathsf{bar{:}0}$, $\mathsf{bar{:}1}$ and $\mathsf{bar{:}2}$; they are mutually exclusive.

Level Grammar. $\Sigma = \mathsf{bar} : 2$, $\Omega = \mathsf{bar} : 0$. There is an axiom $-\mathsf{hd}. \rightarrow .\mathsf{bar} : 2$ and the following rules.



Notice that there is no recursion for $\mathsf{bar{:}0}$. All that needs to be done in order to get a full $\overline{X}$-syntax is to define the set of basic categories. In the barriers system of (Chomsky, 1986), the following set would have sufficed $\{\mathsf{n}, \mathsf{v}, \mathsf{p}, \mathsf{adj}, \mathsf{adv}, \mathsf{i}, \mathsf{c}\}$, but currently there is no observable agreement about the basic set of categories.

SPEC-HEAD-AGREEMENT. In GB, agreement phenomena are assumed to be the effect of *spec-head-agreement.* This means that in a phrase xp the specifier carries the same agreement features as the x^0 head. Suppose, for the sake of the argument, that we want to code spec-head-agreement without assuming that agreement features are head-features (which they are). Then, as the head of a phrase and the specifier are never in the same local tree there is no a priori reason to believe that spec-head-agreement is codable without a memory. Indeed, one can show that a memory is strictly necessary. Just consider the problem of how the head knows

the particular agreement features the specifier has selected if this assigment is not signalled to the x^0-head by its $\bar{\mathsf{x}}$-mother. If the mother agrees with the x^0 as part of the HFC then the head knows what to do; otherwise, if agreement on the mother is independent from that of the x^0-head, we need a memory of sufficient size to transmit the agreement information via the mother to the daughter. With this memory, what one has to do is to transmit the agreement information from specifier to its sister and let it percolate just like a head-feature down to the x^0-head. Instead of this longwinded description we may issue the following convention.
SPEC-HEAD-AGREEMENT. For all agreement terms a

$$\mathsf{hd}. \rightarrow .[\mathsf{a}. \leftrightarrow .((\neg(\overset{\leftarrow}{\Diamond} \cup \overset{\rightarrow}{\Diamond})\mathsf{sp})? \circ \overset{\uparrow}{\Diamond})^* \circ (\overset{\leftarrow}{\Diamond} \cup \overset{\rightarrow}{\Diamond})\mathsf{a}]$$

The costs of coding are calculated as follows. Horizontal modalities are free of cost; likewise $(\neg(\overset{\leftarrow}{\Diamond} \cup \overset{\rightarrow}{\Diamond})\mathsf{sp})?$. Only the upward operator $\overset{\uparrow}{\Diamond}$ in combination with the star introduces the need for a memory. Its size depends on the number of possible agreement situations. Given k atoms for agreement, we need $\kappa = \Gamma 0E70^2\,\mathrm{log}k\urcorner$ basic features as memory. If we add the HFCs, the need for this memory no longer arises and spec-head-agreement is for free.
RELATIVIZED MINIMALITY. According to (Rizzi, 1990) the conditions on movement are subject to a principle called *Relativized Minimality.* Roughly speaking, it says that a trace must be associated with the nearest accessible element in a position which is functionally similar to that of the trace. To make this precise we have to specify first the notion of *functional similarity.* Two positions are **functionally similar** exactly when they are both A-positions or both $\overline{A}$-positions or both (zero-level) heads. [2] To avoid complications we assume that we have a boolean feature arg telling us which node is an A-position and a boolean feature $\mathsf{n\text{-}arg}$ identifying the $\overline{A}$-positions. [3] Next we need to specify *accessibility.* In (Rizzi, 1990), x is **accessible** for y if (i) x c-commands y (ii) no barrier intervenes. Here, **c-command** is the relation $(\overset{\rightarrow}{\Diamond} \cup \overset{\leftarrow}{\Diamond}) \circ \overset{\downarrow}{\Diamond}^*$. A **barrier** is a node not directly selected by a verbal category, thus in first approximation a node not sister to $\mathsf{c} \vee \mathsf{i} \vee \mathsf{v} \vee \mathsf{adj}$ which selects a complement. [4] Again we avoid unnecessary complications by assuming a boolean ba to specify the barriers. We say that between x and y **intervenes a barrier** if there exists a barrier $b > y$ with $\neg\,(x < b)$. The accessibility relation is now specified as

$$(\overset{\leftarrow}{\Diamond} \cup \overset{\rightarrow}{\Diamond}) \cup (\overset{\uparrow}{\Diamond} \circ (\neg\mathsf{ba}\,?))^* \circ (\overset{\leftarrow}{\Diamond} \cup \overset{\rightarrow}{\Diamond})$$

[2] An **A-position** in a projection of a basic category b is a position that is θ-selected by at least one lexical head of category b. So while θ-positions are not fixed in a tree and can vary with the particular lexical entry in that tree, A-positions are fixed independently of the particular lexical entries.

[3] It is not the case that positions which are not A-positions are $\overline{A}$-positions; rather, this division holds of the maximal projections only. Thus the following is valid $\mathsf{arg} \vee \mathsf{n\text{-}arg}. \leftrightarrow .\mathsf{bar{:}2}$ and, of course, $\mathsf{arg} \wedge \mathsf{n\text{-}arg}. \leftrightarrow .0.$

[4] We simplify matters here by not considering possible complications due to adjunction.

Moreover, immediate sisters are excluded so that the right form of relation is

$$acc = (\overset{\uparrow}{\Diamond} \circ (\neg \mathsf{ba}\,?))^* \circ (\overset{\leftarrow}{\Diamond} \cup \overset{\rightarrow}{\Diamond})$$

Finally, we have as an effect of the general principle of Relativized Minimality several axioms; for example, that an np-trace in A-position needs an accessible antecedent

Relativized Minimality (np-argument trace).

$$\mathsf{np} \wedge \mathsf{arg} \wedge \mathsf{trace}. \rightarrow .acc\ (\mathsf{np} \wedge \mathsf{arg})$$

In the language of GPSG we are allowed to use variables for categories. So we can summarize the full principle by

Relativized Minimality (domain based).

$$\mathsf{trace} \wedge \mathsf{arg} : \beta \wedge \mathsf{head} : \alpha. \rightarrow .acc\ (\mathsf{head} : \alpha \wedge \mathsf{arg} : \beta)$$

Here α ranges over head-features and β over $\{+, 0, -\}$ with $[\mathsf{arg} : +]$ representing arg, $[\mathsf{arg} : -]$ representing n-arg and $[\mathsf{arg} : 0]$ collecting the rest, i. e. the non-phrasal nodes.

The following important point needs to be considered. Up to now we have secured the existence of an accessible antecedent; but it is possible that an antecendent is accessible from several positions. Since the idea behind movement is that only one of the accessing positions can be the origin of the constituent in question, GB uses indices (among other things) to mark those elements which are in a movement chain of a single constituent from its D-structure position to its S-structure position (LF-position). In our language there is no room for indices and one may wonder whether we need them at all for syntactic purposes. GPSG has shown with a particular example that we might get away without them. Here we will show that nothing in (Rizzi, 1990) forces us to assume indices. The problem in the sketch above is that it is consistent to have two nodes y_1, y_2 for which x is accessible and functionally similar. We need to exclude this. Two solutions offer themselves. The first is to specify a relation *co-acc* which holds of nodes y_1 and y_2 if they are functionally similar but different and are *acc*-related to a functionally similar x. Then add the following axiom

Unique Association.

$$\mathsf{trace} \wedge \mathsf{head} : \alpha \wedge \mathsf{arg} : \beta. \rightarrow . \neg co\text{-}acc\ (\mathsf{trace} \wedge \mathsf{head} : \alpha \wedge \mathsf{arg} : \beta)$$

The second solution is to monitor directly the distribution of the traces that can be associated with an antecedent. To this end introduce two (!) type 1 features, acc and p-acc with range $\{\mathsf{head}, \mathsf{arg}\}$ (corresponding to finitely many boolean atoms!). Let $\mathsf{p\text{-}acc} : \alpha$ be initiated on a node exactly when it has a sister of type α. $\mathsf{p\text{-}acc} : \alpha$ is passed on as $\mathsf{acc} : \alpha$ to the daughters exactly as is $\mathsf{acc} : \alpha$. $\mathsf{acc} : \alpha$ percolates downwards from mother to at most one daughter, all others get $\neg(\mathsf{acc} : \alpha)$. If the mother is a barrier it fails to pass on its acc-feature; the value of the acc-feature on the daughter then depends on the presence of an antecedent among the daughters themselves. The complication in the introduction of p-acc lies in the fact that sisters of ancedents cannot access them via *acc* – only their daughters and offsprings can. The following then expresses Relativized Minimality with Unique Association.

Relativized Minimality (coded).

$$\mathsf{trace} \wedge \mathsf{head} : \alpha \wedge \mathsf{arg} : \beta . \to .\mathsf{acc} :[\mathsf{head} : \alpha \wedge \mathsf{arg} : \beta]$$

Finally, let me say that this type of approach is only possible under the condition that subsequent derivations do not destroy the configuration that justified the previous derivation. This is the case. Head-movement is an exception, however. For example, if v adjoins to infl and the complex [v infl] adjoins to comp then after the second adjunction the link between the first trace and its antecedent is broken and the resulting structure fails the condition on head movement for that trace. For these cases a different approach is called for. [5]

7. Conclusion

The Coding Theorem and the machinery surrounding it can be used to analyze a wide variety of syntactic theories. I have exemplified this with GPSG and some variant of GB. It turned out that feature grammars provide just the right kind of object with which to compare the two prima facie quite distinct formalisms. Many questions then arise. First, is it possible to collapse the different levels of representation of GB into a single one in a similar mechanical way as with syntactic codes, that is, is there a way to reduce GB to an essentially monostratal theory? If so, the main obstacle in comparing GB with other theories is removed. Secondly, we have seen that grammars can also be equated with axiomatic theories, via the characteristic axiom. If a full GB-theory is treated this way this characteristic axiom cannot be constant in case the generated language is not context-free. Thus, for any suitable pair of syntactic theories is it possible to compare them just by looking at the characteristic axiom? A first step has been made. If this axiom is equivalent to a constant axiom then by the Coding Theorem the grammar generates a context-free language. However, what happens in other cases is largely unexplored. It might be worthwile to investigate whether we can measure the complexity problems of the language (e. g. the parsing problem) via the characteristic axiom analogous to descriptive complexity theory.

In sum, this may be a way to replace the by now standard approach to formal language theory via rule systems by another which is more in line with the current

[5] This is in essence the example with which Chomsky demonstrates the difference between purely representational accounts of movement and a derivational one.

developments in linguistics. As the rules move out of focus and get replaced by principles we need ways to analyse the content of a theory phrased not with rules but with principles. At the same time it would be unwise to just start from scratch. Rather, one wants a way to connect one way of thinking with the other. Coding Theory is a way to do that.

References

Noam Chomsky. *Barriers.* MIT Press, Cambrigde (Mass.), 1986.

B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* Camnridge University Press, Cambridge, 2 edition, 1991.

Gerald Gazdar, Ewan Klein, Geoffrey Pullum, and Ivan Sag. *Generalized Phrase Structure Grammar.* Blackwell, Oxford, 1985.

D. Harel. Dynamic logic. In Dov M. Gabbay and Franz Guenthner, editors, *Handbook of Philosophical Logic*, volume 2. Reidel, 1984.

Michael A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, Reading (Mass.), 1978.

Marcus Kracht. Mathematical aspects of command relations. In *Proceedings of the EACL 93*, pages 240 – 249, 1993.

Luigi Rizzi. *Relativized Minimality.* MIT Press, Boston (Mass.), 1990.

Edward P. Stabler. *The Logical Approach to Syntax. Foundation, Specification and Implementation of Theories of Government and Binding.* ACL-MIT Press Series in Natural Language Processing. MIT Press, Cambridge (Mass.), 1992.